# Retrieval Models for Text Documents

- CS 293S, 2022. Tao Yang

# Outline

- **Overview**
  - Which results satisfy the query constraint?
  - Focus on text documents with a flat structure
    - Web page retrieval can use more structural features.
- **Boolean model**
  - Document processing steps
  - Query processing
- **Statistical vector space model**
- **Neural representations with word embeddings**
- **Neural representations with pretrained language models**
  - BERT (next time)

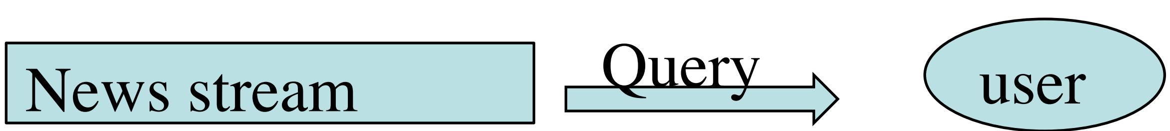
# Document Retrieval/Ranking Applications

| Task | X | Y |
|---|---|---|
| Ad-hoc retrieval | Query | Document |
| Filtering | Fixed query or profile | Document streams |
| Question-answering | Question | Answer |
| Automatic conversation | Dialog | Response |

**Ad-hoc document retrieval**:

- Query is typically short and keyword based;
- Documents can vary considerably in length, from tens of words to thousands

**Filtering**



# Retrieval Models and Tasks

- **A retrieval model specifies the details of:**
  - 1) Document representation. Query representation
  - 2) Retrieval function: how to find relevant results
  - 3) Determines a notion of relevance.
    - Imply the order of ranking
- **Classical models**
  - Boolean models
  - Vector space models
  - Probabilistic models
- **Neural models**
  - Word embedding
  - Pretrained language models
    - Contextual embeddings

# Boolean Model

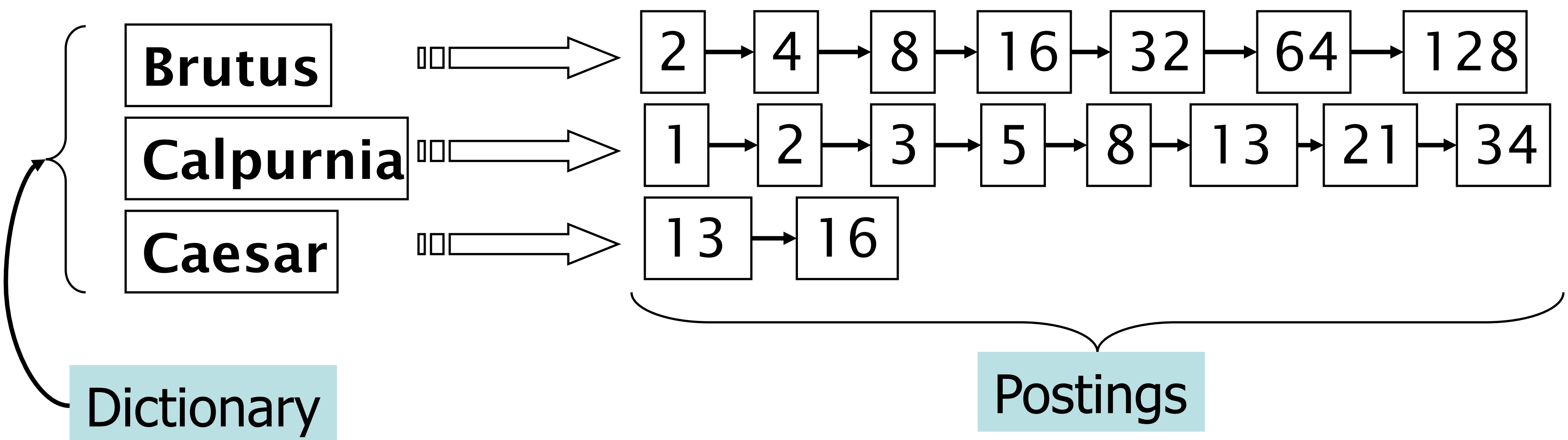
- A document is represented as a set of keywords.
- Queries are Boolean expressions of keywords, connected by AND, OR, and NOT, including the use of brackets to indicate scope.
  - Rio & Brazil | Hilo & Hawaii,      hotel & !Hilton
- Output: Document is relevant or not. No partial matches
- Example for Shakespeare plays with a bit vector representation

1 if play contains word, 0 otherwise

| | Antony and Cleopatra | Julius Caesar | The Tempest | Hamlet | Othello | Macbeth |
|---|---|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 | 0 | 1 |
| Brutus | 1 | 1 | 0 | 1 | 0 | 0 |
| Caesar | 1 | 1 | 0 | 1 | 1 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 | 0 | 0 |

# Inverted index: Sparse vectors with list representation

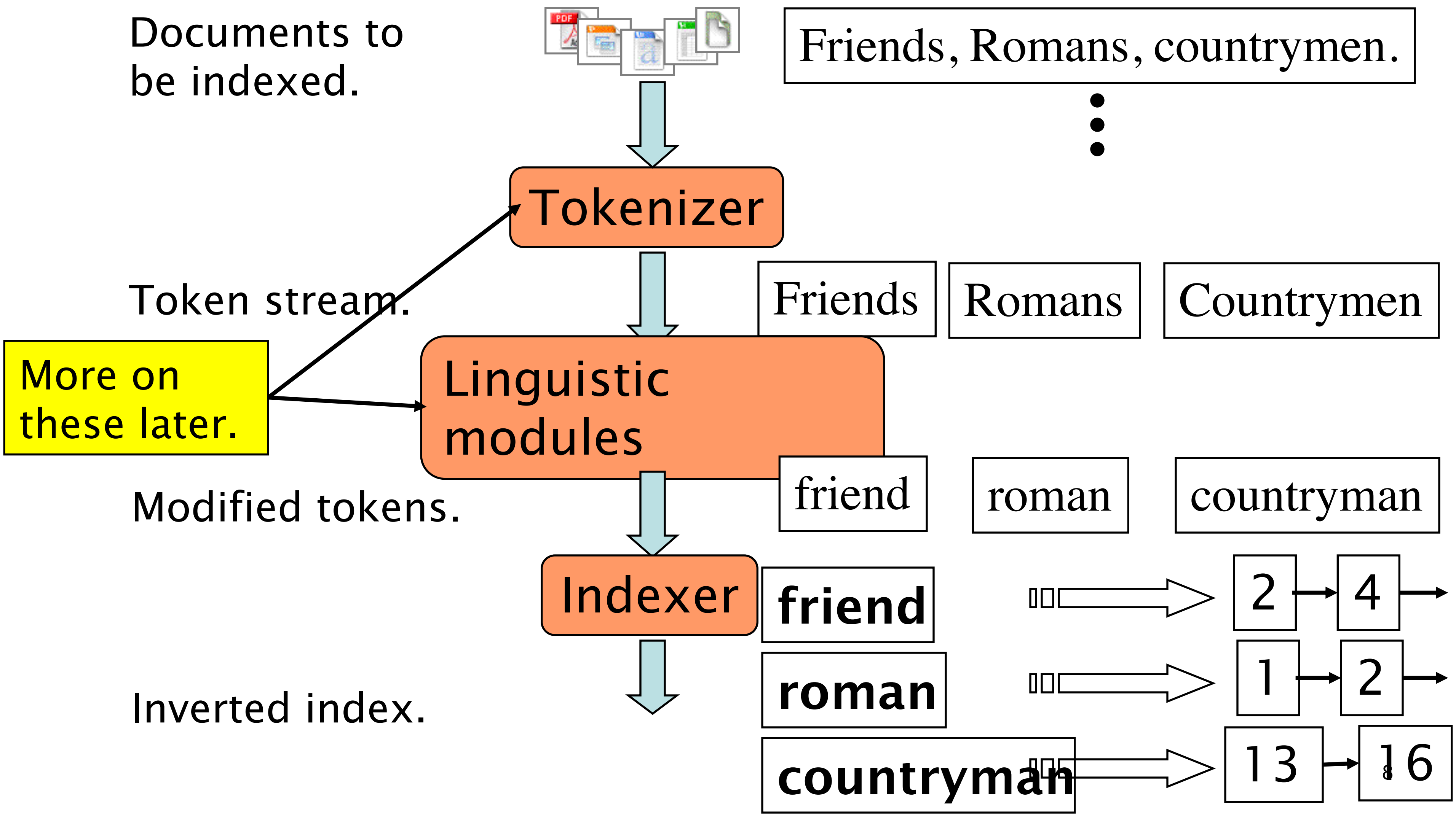
- **Incident vectors are sparse → sparse matrix**
  - Compact representation needed  to save storage
- **Inverted index with list representation**
  - For each term *T*, must store a list of all documents that contain *T*.



# Document Preprocessing Steps

- **Strip unwanted characters/markup** (e.g. HTML tags, punctuation, numbers, etc.).
- **Break into tokens (keywords) on whitespace.**
- **Possible linguistic processing** (used in some applications, but dangerous for general web search)
  - Stemming (cards ->card)
  - Remove common stopwords (e.g. a, the, it, etc.).
  - Used sometime, but dangerous
- **Build inverted index**
  - keyword → list of docs containing it.
  - Common phrases may be detected first using a domain specific dictionary.

# Inverted index construction



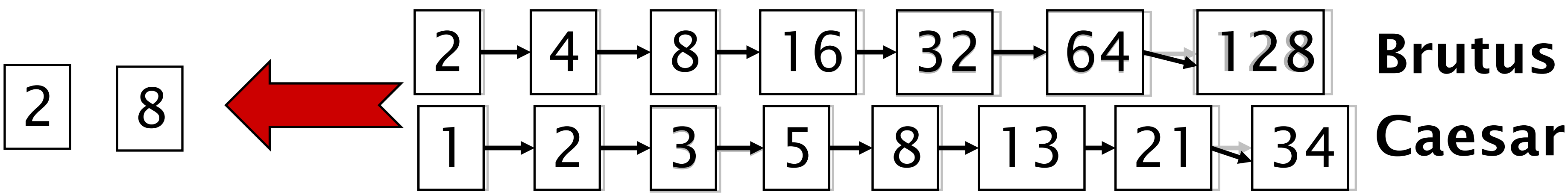
# Discussions

- **Which terms in a doc do we index?**
  - All words or only "important" ones?
- **<u>Stopword</u> list: terms that are so common**
  - they MAY BE ignored for indexing.
  - *e.g.*, ***the, a, an, of, to*** …
  - language-specific.
- **How do we process a query?**
  - What kinds of queries can we process?

# Query processing

- **Consider processing the query:**

  ***Brutus*** *AND* ***Caesar***
  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate *Caesar* in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:




If the list lengths are m and n, the merge for sorted lists takes O(m+n) operations.

Crucial: postings sorted by docID.

# Boolean Models – Problems

- **Very rigid: AND means all; OR means any.**
  - Easy to understand. Clean formalism.
- **Difficult to express complex user requests.**
  - Still too complex for general web users
- **Difficult to control the number of documents retrieved.**
  - *All* matched documents will be returned.
- **Difficult to rank output.**
  - *All* matched documents logically satisfy the query.
- **Difficult to perform relevance feedback.**
  - If a document is identified by the user as relevant or irrelevant, how should the query be modified?

# Example Application: WestLaw

http://www.westlaw.com/

- **Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)**
  - Long, precise queries; proximity operators; incrementally developed; not like web search
  - Professional searchers (e.g., Lawyers) still like Boolean queries: You know exactly what you're getting.
- **Example query with proximity operators:**
  - What is the statute of limitations in cases involving the federal tort claims act?
  - LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM

# Outline

- **Overview**
  - Which results satisfy the query constraint?
- **Boolean model**
  - Document processing steps
  - Query processing
- **Statistical vector space model**
- **Neural representations with word embeddings**
- **Neural representations with pretrained language models**
  - BERT (next time)

# Statistical Vector Space Model

- A document is typically represented by a *bag of words* (unordered words with frequencies).
- Bag = set that allows multiple occurrences of the same element.
- User specifies a set of desired terms with optional weights:
  - Weighted query terms:

    Q = < database 0.5; text 0.8; information 0.2 >
  - Unweighted query terms:

    Q = < database; text; information >
  - No Boolean conditions specified in the query.
- Retrieval based on *similarity* between query and documents.
  - Output documents are ranked by similarity to query.
- Weights in vectors
  - Similarity based on occurrence *frequencies* of keywords in query and document.

# The Vector-Space Representation

- Assume *t* distinct terms remain after preprocessing; call them index terms or the vocabulary.
- Each term, *i*, in a document or query, *j*, is given a real-valued weight, $w_{ij}$.
- Both documents and queries are expressed as t-dimensional vectors:

$$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj})$$

$$\begin{array}{c} \\ D_1 \\ D_2 \\ : \\ : \\ D_n \end{array} \begin{array}{c} \begin{array}{cccc} T_1 & T_2 & \ldots. & T_t \end{array} \\ \left(\begin{array}{cccc} w_{11} & w_{21} & \ldots & w_{t1} \\ w_{12} & w_{22} & \ldots & w_{t2} \\ : & : & & : \\ : & : & & : \\ w_{1n} & w_{2n} & \ldots & w_{tn} \end{array}\right) \end{array}$$

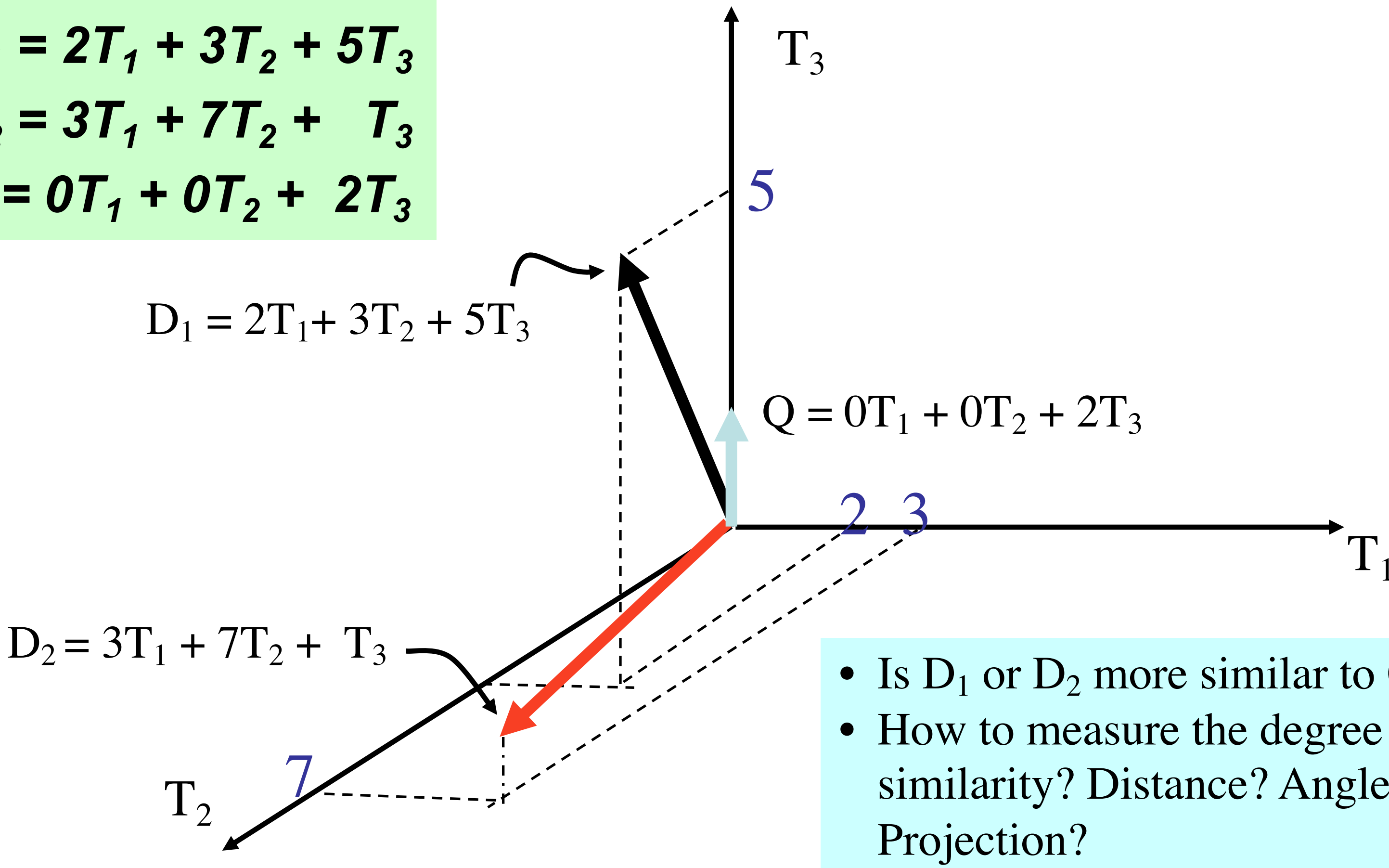
# Example: Graphic representation

**Example*:***

***$D_1 = 2T_1 + 3T_2 + 5T_3$***

***$D_2 = 3T_1 + 7T_2 + T_3$***

***$Q = 0T_1 + 0T_2 + 2T_3$***




- Is $D_1$ or $D_2$ more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

# Issues for Vector Space Model

- How to determine important words in a document?
  - Word n-grams (and phrases, idioms,…) → terms
- How to determine the degree of importance of a term within a document and within the entire collection?
- How to determine the degree of similarity between a document and the query?
- In the case of the web, what is a collection and what are the effects of links, formatting information, etc.?

# Term Weights: Term Frequency

- More frequent terms in a document are more important, i.e. more indicative of the topic.

  $f_{ij}$ = frequency of term $i$ in document $j$

- May want to normalize *term frequency* (*tf*) across the entire corpus:

  $tf_{ij} = f_{ij} \; / \; max\{f_{ij}\}$

- Terms that appear in many *different* documents are *less* indicative of overall topic. *Less discrimination* power.

  $df_i$ = document frequency of term $i$

  = number of documents containing term $i$

  $idf_i$ = **inverse document frequency** of term $i$,

  $= \log_2 (N/ df_i)$ N: total number of documents

- Log used to dampen the effect relative to *tf*.

# TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij}\, idf_i = tf_{ij} \log_2 (N/\, df_i)$$

- A term occurring frequently in the document but rarely in the rest of the collection is given high weight.

Example: A document has term frequencies: A(3), B(2), C(1)

Assume collection contains 10,000 documents and

document frequencies of these terms are: A(50), B(1300), C(250)

Then:

A: tf = 3/3; idf = log(10000/50) = 5.3; tf-idf = 5.3

B: tf = 2/3; idf = log(10000/1300) = 2.0; tf-idf = 1.3

C: tf = 1/3; idf = log(10000/250) = 3.7; tf-idf = 1.2

# Similarity Measure

- A similarity measure is a function that computes the *degree of similarity* between two vectors.
- Using a similarity measure between the query and each document:
- Similarity between vectors for the document $d_i$ and query $q$ can be computed as the vector inner product:

$$\mathrm{sim}(d_j, q) = d_j \bullet q = \text{sum} \quad w_{ij} \cdot w_{iq}$$

where $w_{ij}$ is the weight of term $i$ in document $j$ and $w_{iq}$ is the weight of term $i$ in the query

**Example:**

| | database | retrieval | architecture | computer | information | management | text |
|---|---|---|---|---|---|---|---|
| D = | 1, | 1, | 1, | 0, | 1, | 1, | 0 |
| Q = | 1, | 0 , | 1, | 0, | 0, | 1, | 1 |

sim(D, Q) = 3

# Example & Properties of Inner Product

Another example with weighted vectors:

$D_1 = 2T_1 + 3T_2 + 5T_3 \qquad D_2 = 3T_1 + 7T_2 + 1T_3$

$Q = 0T_1 + 0T_2 + 2T_3$

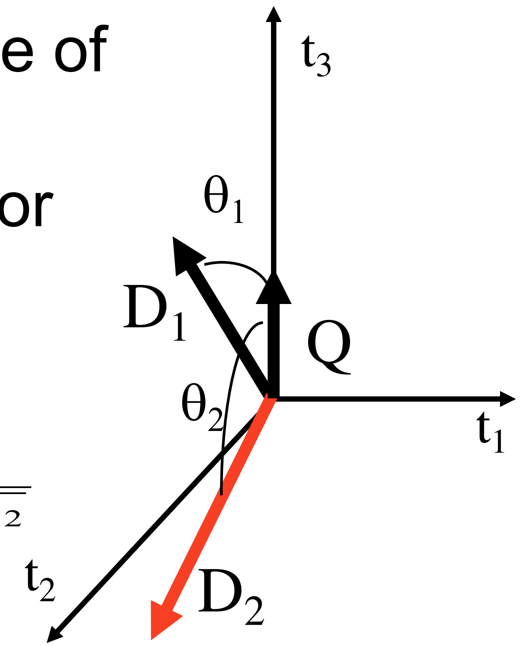
$sim(D_1, Q) = 2*0 + 3*0 + 5*2 = 10$

$sim(D_2, Q) = 3*0 + 7*0 + 1*2 = 2$

- **Properties of Inner Product**
  - The inner product is unbounded.
  - Favors long documents with a large number of unique terms.
  - Measures how many terms matched but not how many terms are *not* matched.

# Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.




$$\text{CosSim}(\mathbf{d}_j, \mathbf{q}) = \frac{\vec{d_j} \bullet \vec{q}}{\left|\vec{d_j}\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t} (w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$

$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim($D_1$ , Q) = 10 / $\sqrt{(4+9+25)(0+0+4)}$ = 0.81
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim($D_2$ , Q) = 2 / $\sqrt{(9+49+1)(0+0+4)}$ = 0.13
$Q = 0T_1 + 0T_2 + 2T_3$

$D_1$ is 6 times better than $D_2$ using cosine similarity but only 5 times better using inner product.

# Improvement: BM25

Document component: term frequency

Query component

- Rank or feature score with an extension of TF-IDF
- Given document d for query q

$$\sum_{w \in q \cap d} \ln \frac{N - df(w) + 0.5}{df(w) + 0.5} \cdot \frac{(k_1 + 1) \times c(w,d)}{k_1((1-b) + b\frac{|d|}{avdl}) + c(w,d)} \cdot \frac{(k_3+1) \times c(w,q)}{k_3 + c(w,q)}$$

IDF

Scaled document length

- df: nuber of documents containing this word
- |d|: document length
- avdl: average document length
- c(w,d): term frequency in this document
- c(w,q): term frequency in this query
- Constants $k_1$ in [1,2], b=0.75, $k_3$ in [0, 3000]

# Comments on Vector Space Models

- **Simple, practical, and mathematically based approach**
- **Provides partial matching and ranked results.**
- **Problems**
  - Missing syntactic information (e.g. phrase structure, word order, proximity information).
  - Missing semantic information
    - word sense: multiple meanings of a word
    - Assumption of term independence. ignores synonymy.
  - Lacks the control of a Boolean model (e.g., *requiring* a term to appear in a document).
    - Given a two-term query “A B”, may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.

# Outline

- **Overview**
  - Which results satisfy the query constraint?
- **Boolean model**
  - Document processing steps
  - Query processing
- **Statistical vector space model**
- **Neural representations with word embeddings**
- **Neural representations with pretrained language models**
  - BERT (next time)

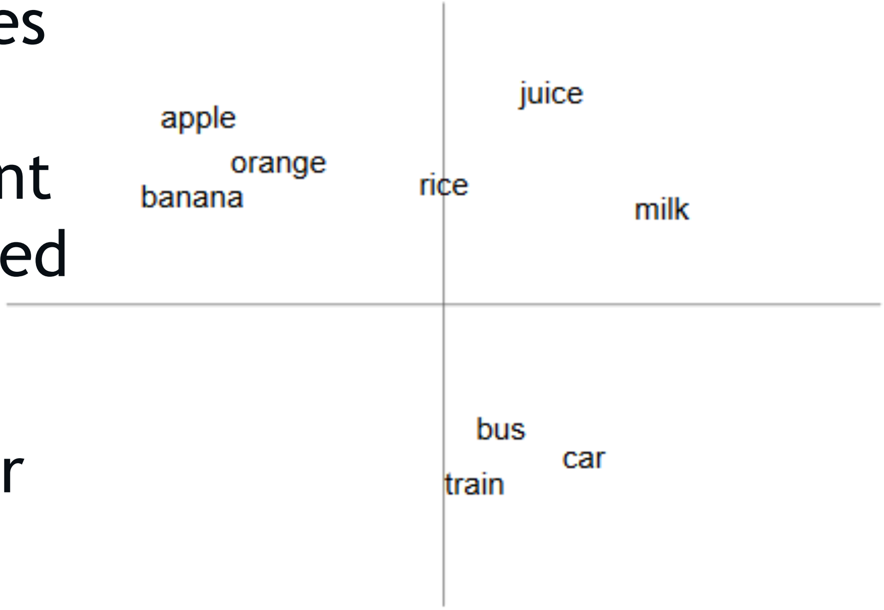
# Word Representations

| Traditional Method - Bag of Words Model | Word Embeddings |
|---|---|
| • Uses one hot encoding<br><br>• Each word in the vocabulary is represented by one bit position in a HUGE vector.<br><br>• For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . . . 0 0 0 0<br><br>• Context information is not utilized | • Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300)<br><br>• Unsupervised, built just by reading huge corpus<br><br>• For example, "Hello" might be represented as :<br>[0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] |

# Word embedding: Motivation for a new word representation

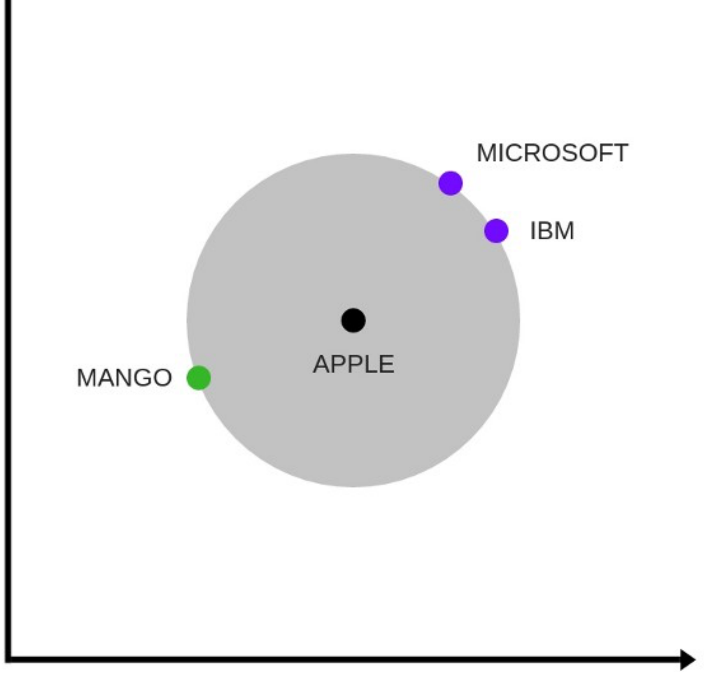
A Word Embedding format generally tries to map a word to a numerical vector.

- A representation that captures words' *meanings*, *semantic relationships* and the different types of contexts they are used in
- Similar words tend to occur together and will have similar context– Orange is a fruit. Banana is a fruit. They have a similar context i.e fruit.




- A context may be a single word or a group of words.

# Usage of Word Embeddings




- Similarity distance of mango, apple, Microsoft, IBM

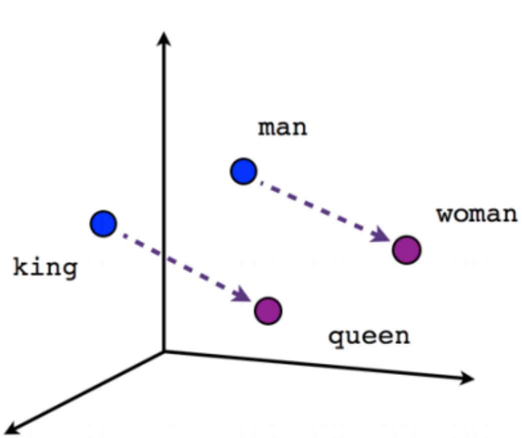
- Finding the degree of similarity between two words.
similarity('woman','man')= 0.73723527

- Finding odd one out.
doesnt_match('breakfast cereal dinner lunch') = 'cereal'

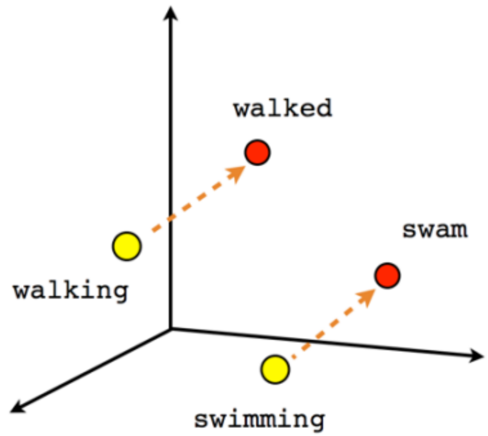
- Compute woman+king-man =queen
most_similar(positive=['woman','king'],negative=['man'])
queen: 0.508

# Examples on Characteristics of Word Embeddings

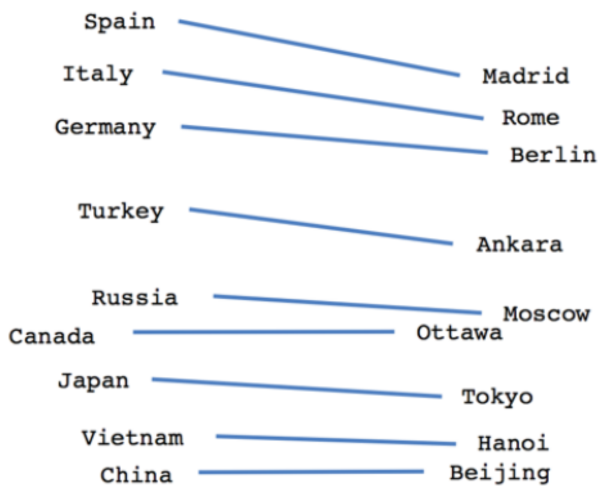
Numerical representations of contextual similarities between words




Male-Female




Verb tense




Country-Capital

vector[Queen] = vector[King] - vector[Man] + vector[Woman]

# Data and Software for word2vec

- Easiest way to use it is via the Gensim libarary for Python (tends to be slowish, even though it tries to use C optimizations like Cython, NumPy)

https://radimrehurek.com/gensim/models/word2vec.html

- Original word2vec C code by Google

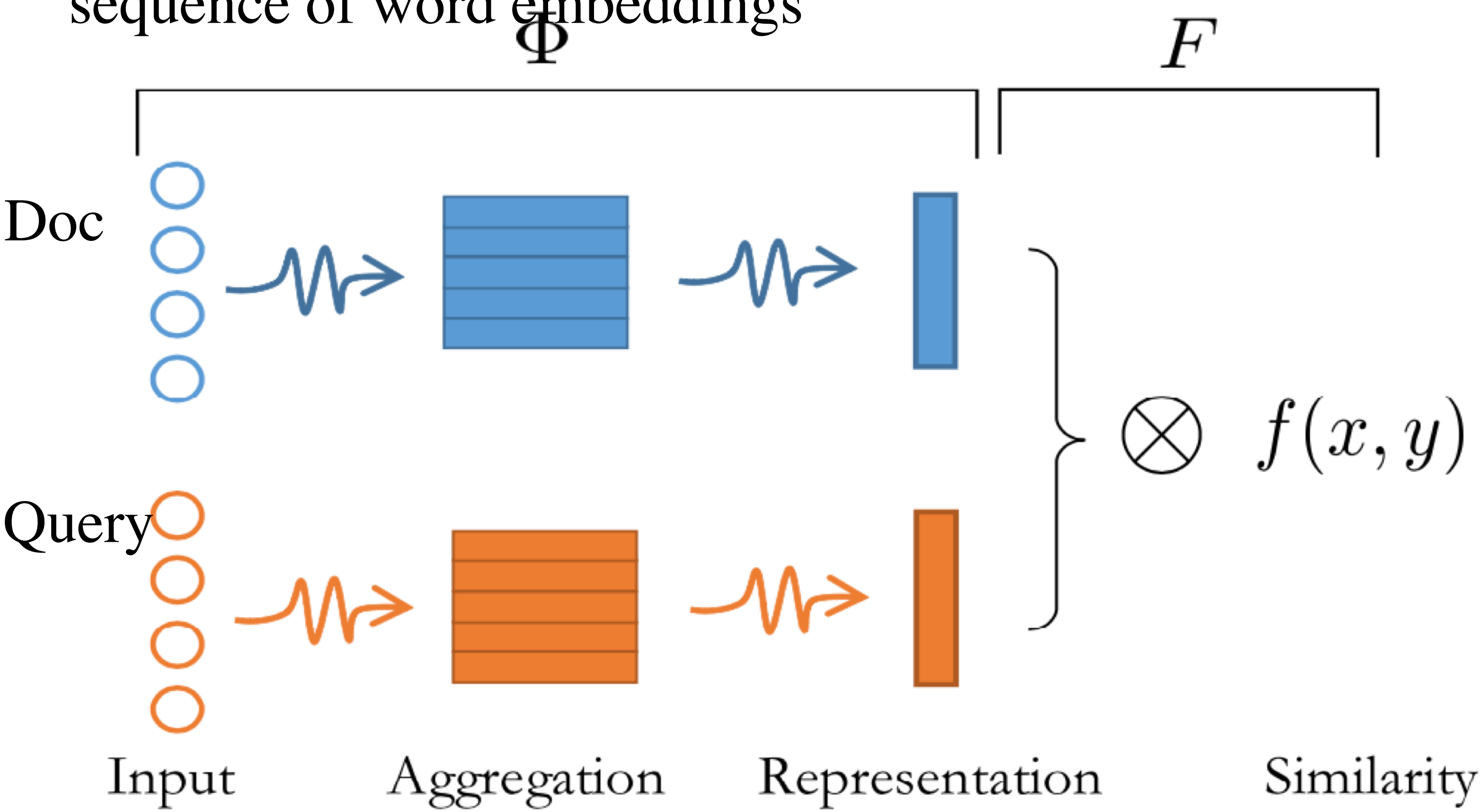
https://code.google.com/archive/p/word2vec/

# Use of word embedding in document matching: Representation-based neural ranking

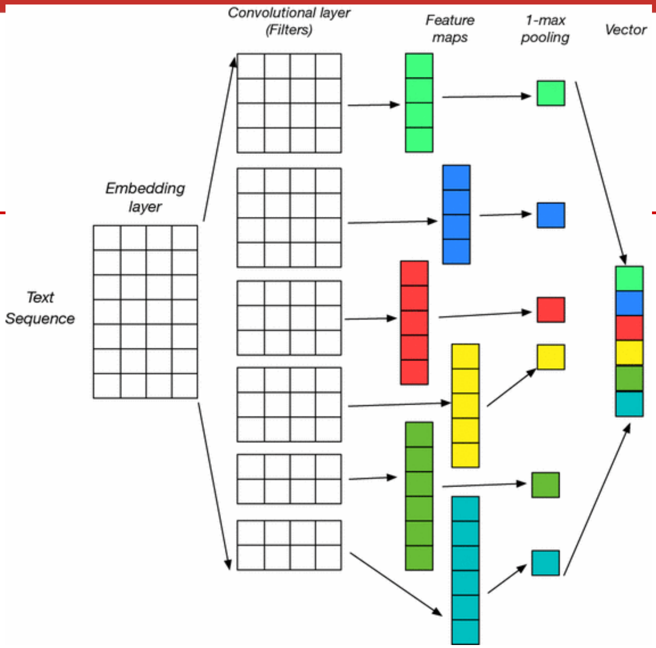
Match(query,doc)=F(Φ(query),Φ(doc))

F: scoring function

Φ: map to a document representation vector with a sequence of word embeddings



# Neural Information Retrieval




## GROWING PUBLICATION POPULARITY AT TOP IR CONFERENCES

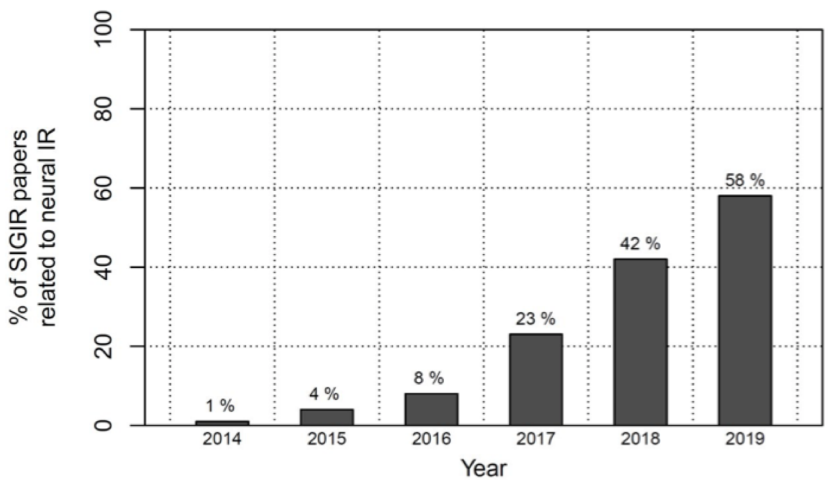



**Figure 1.1:** The percentage of neural IR papers at the ACM SIGIR conference—as determined by a manual inspection of the papers—shows a clear trend in the growing popularity of the field.

Bhaskar Mitra, 2019

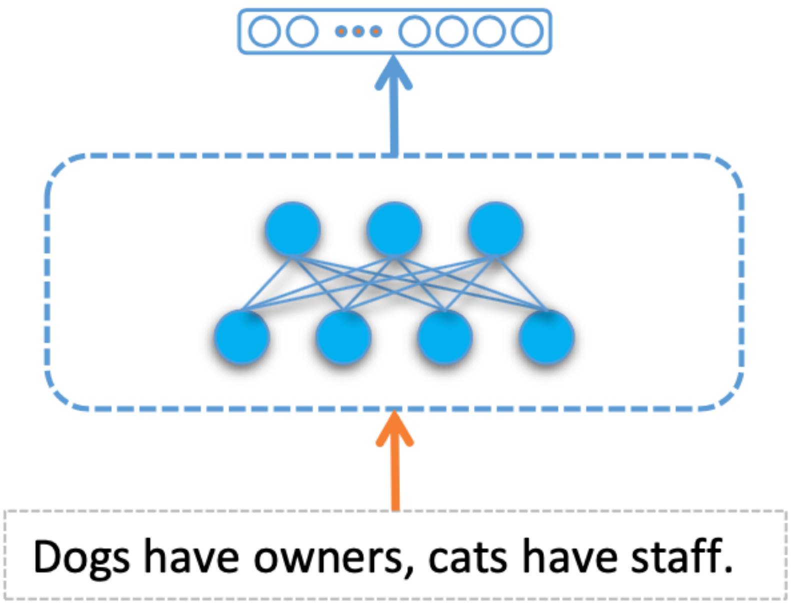
## STRONG PERFORMANCE AGAINST TRADITIONAL METHODS IN TREC 2019


(a) Document retrieval task

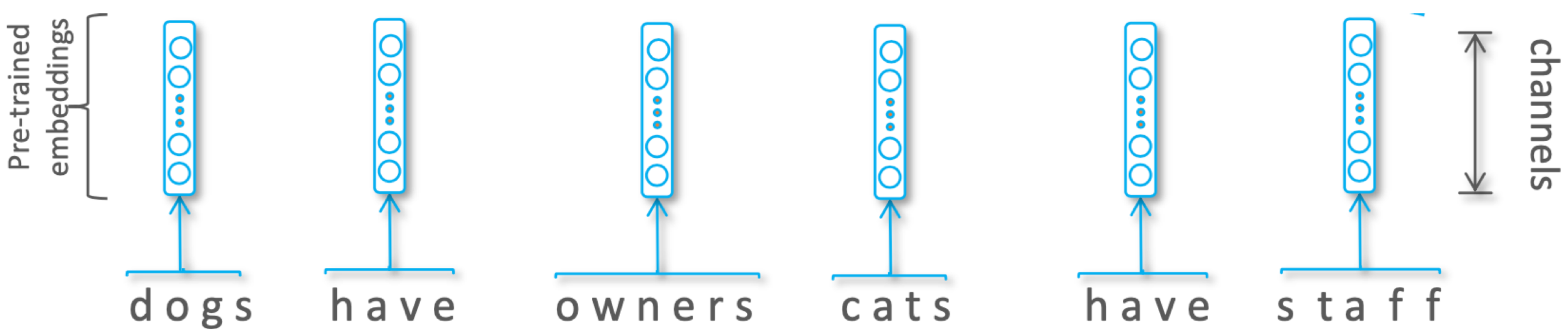
Figure 1: NDCG@10 results, broken down by run type. Runs of type "nnlm", meaning they use language models such as BERT, performed best on both tasks. Other neural network models "nn" and non-neural models "trad" had relatively lower performance this year. More iterations of evaluation and analysis would be needed to determine if this is a general result, but it is a strong start for the argument that deep learning methods may take over from traditional

# Neural Representation of Documents and Queries with Word Embeddings

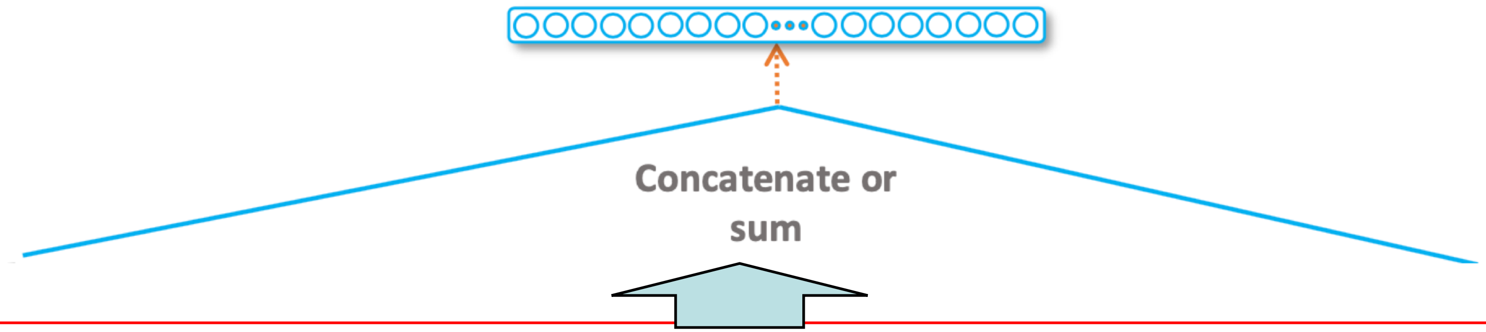
- What to feed to a neural network given a document or query?



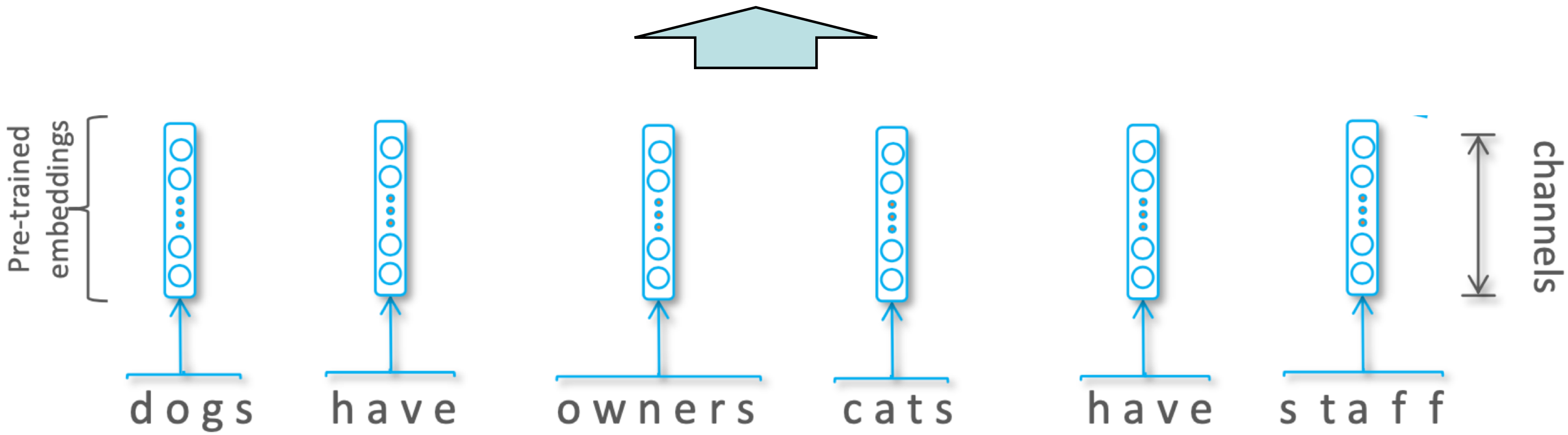

- Use pretrained word embeddings:



# Further Processing of Neural Input Vectors



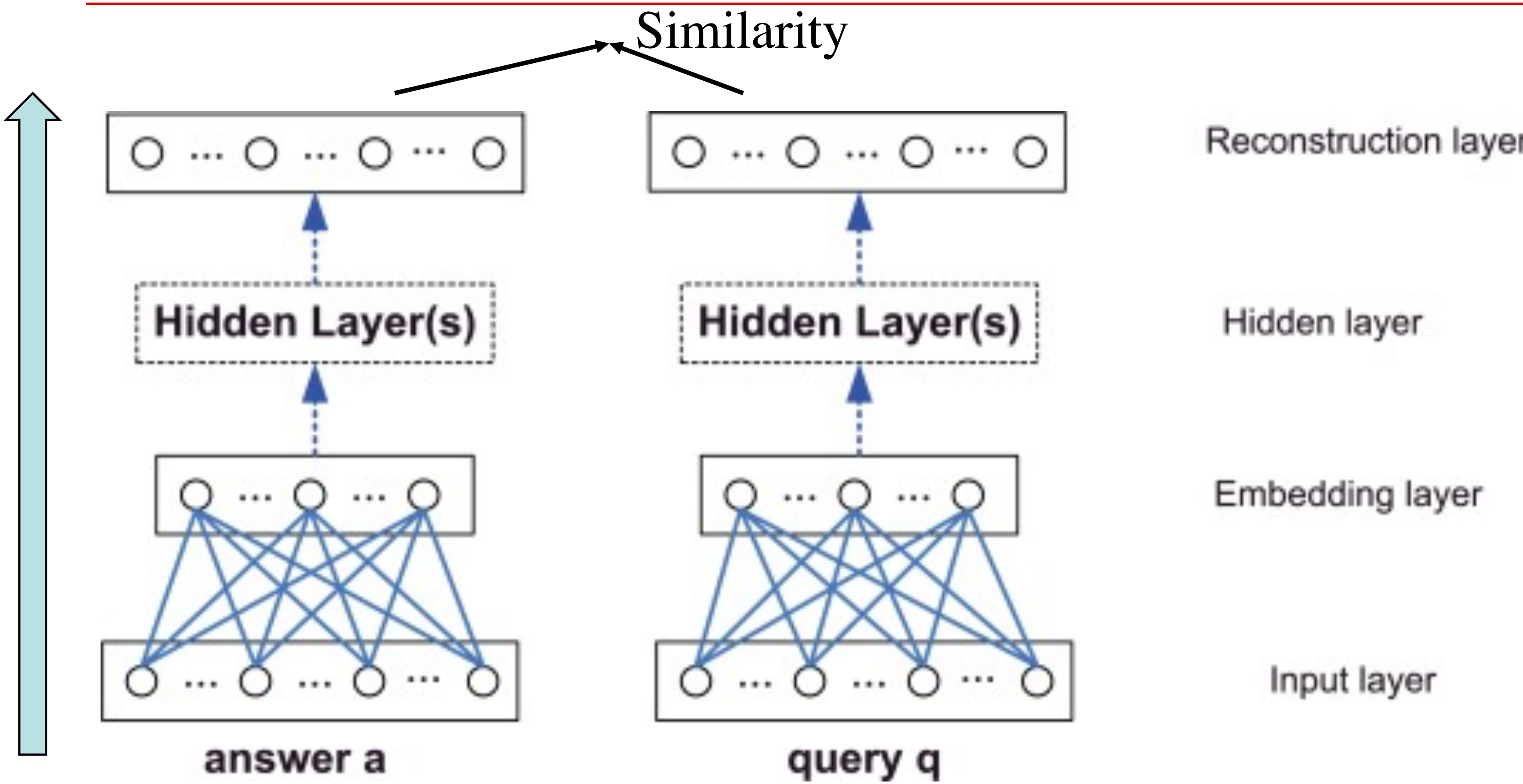

Additional transformation such as convolution network. Think about it as matrix-matrix multiplication



# Categories of Neural Ranking with Word Embeddings

- **Representation-focused models**
  - Query is converted with a neural representation and transformation
  - Document is converted with a neural representation and transformation
  - Finally query-document similarity is computed.
  - Useful for long/NLP queries
- **Interaction focused model**
  - The pairwise interaction of query terms and document terms is computed first
  - The interaction result is converted with a neural network
  - Effective for short keyword queries

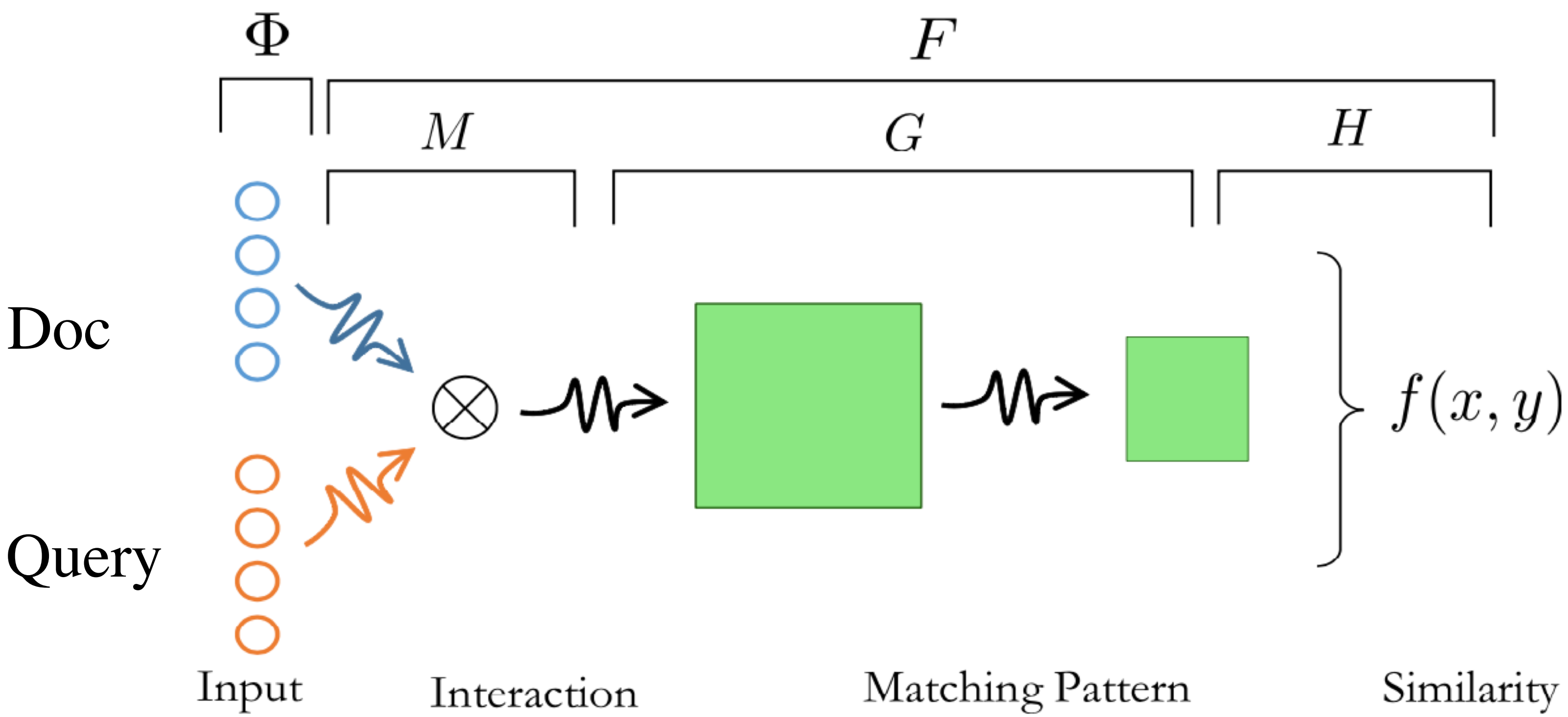
# Example of Representation-focused models




DSSM [Huang et al. CIKM 2013]
C-DSSM [Shen et al. WWW 2014]
ARC-I [Hu et al. NIPS 2014]

# 2nd Category of Neural Ranking

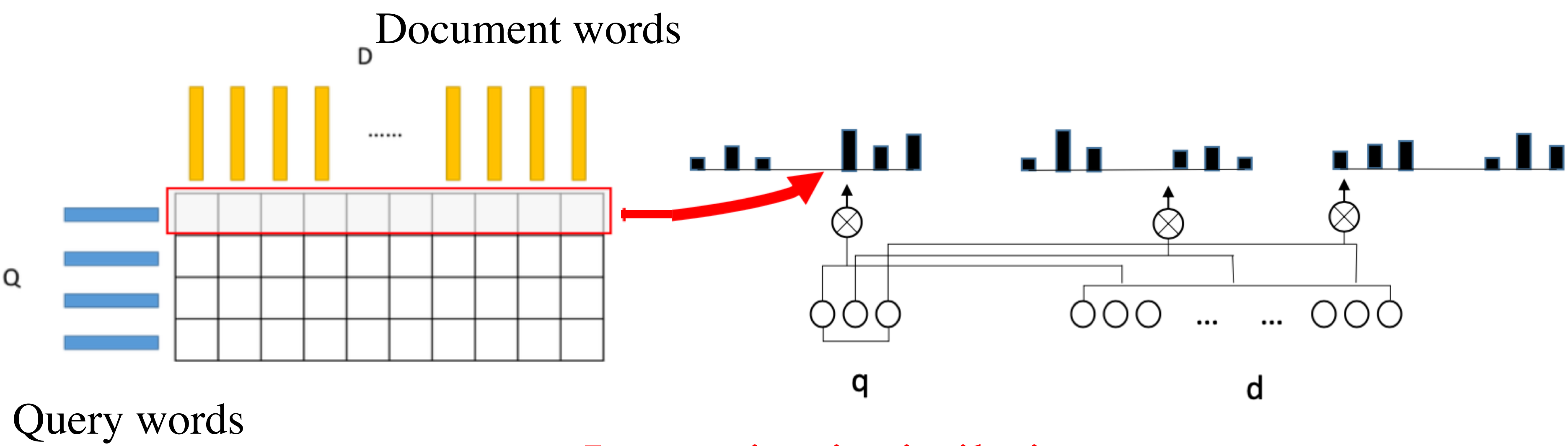
- The pairwise interaction of query terms and document terms is computed first
- The interaction result is converted with a neural network.




Match(query,doc)=F(Φ(query),Φ(doc))

# Example of Interaction-focused Ranking: DRMM

- J. Guo, Y. Fan, Q. Ai, and W.B. Croft. A deep relevance matching model for ad-hoc retrieval. CIKM 2016.




Interaction is similarity histogram distribution between each query word with a set of words in a document

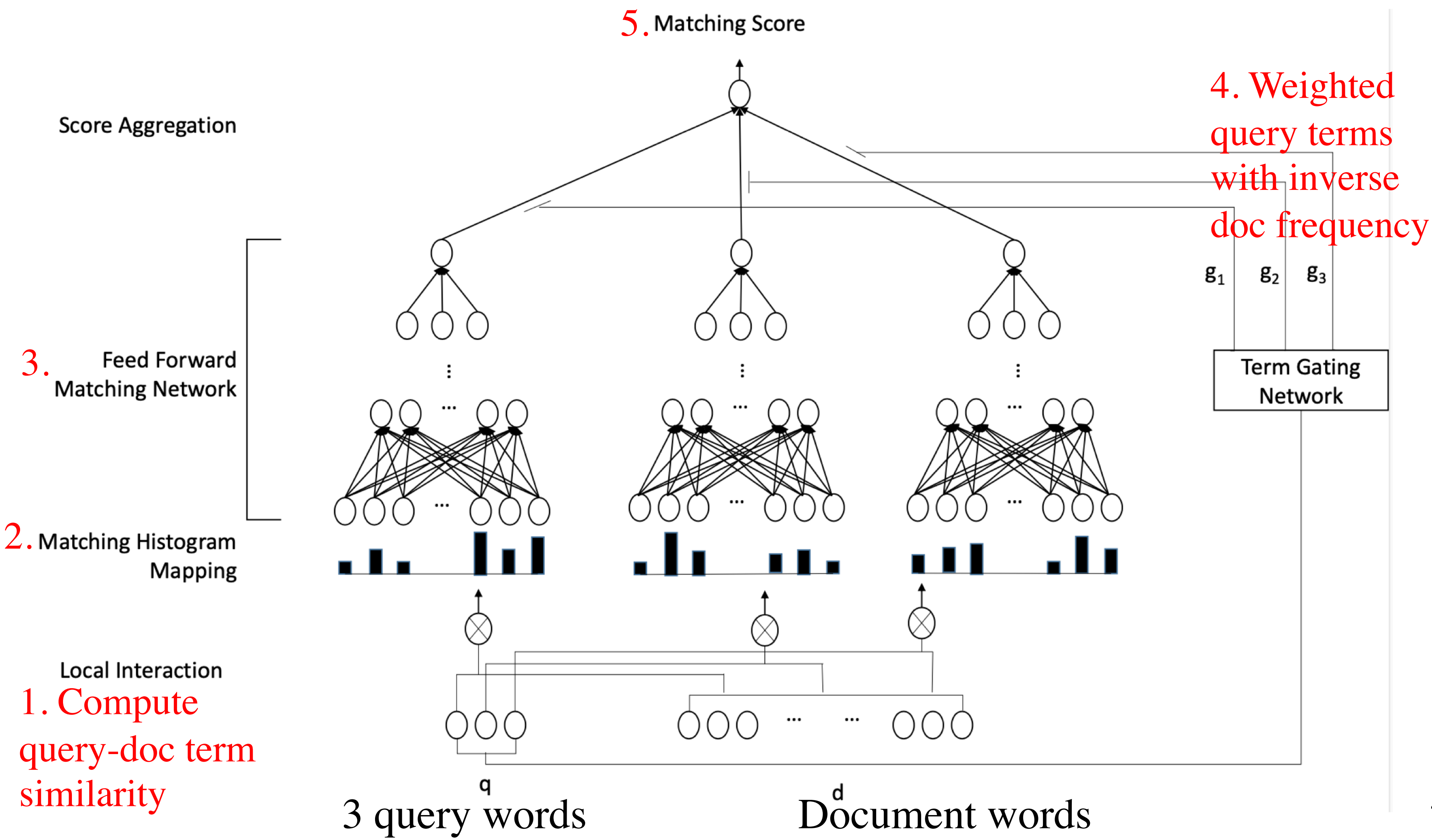
# Example of DRMM in computing similarity histogram distribution

- **Query:** car rental deal
- **Document** ={car, rent, truck, bump, injunction, runway}

- Compute the **interaction of "car"** with this document
  - Use word2vec emeddings to compute the cosine similarity of "car" with each word in this document.
    - **(1, 0.2, 0.7, 0.3, −0.1, 0.1)**
  - Derive **a histogram distribution** with five similarity bins
    - {[−1, −0.5), [−0.5, −0), [0, 0.5), [0.5, 1), [1, 1]}
    - Distribution for this example: **[0, 1, 3, 1, 1]**
    - Logarithm may be applied to the histogram count
- Further compute the **interaction of "rental", and "deal"** separately with this document to get another histogram vector
- Use a forward network to combine these histogram vectors with query term weights, and report a final rank score.

# Processing Flow of Neural Ranking in DRMM

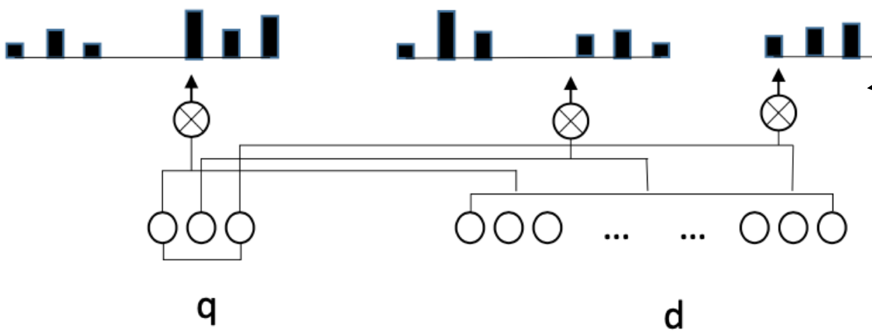
## Flow for a query with 3 words



# Steps 1, 2 and Step 3 of DRMM

Cosine similarity

- **Steps 1& 2:** Similarity-based histogram computation




$$z_i^{(0)} = h\left(w_i^{(q)} \otimes d\right)$$

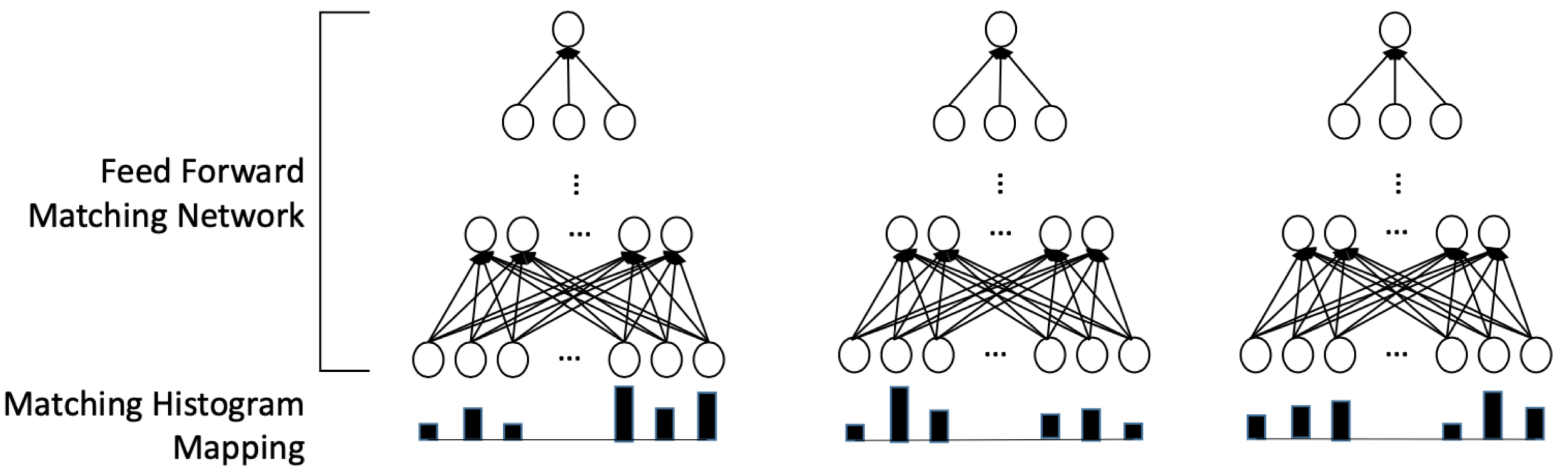
i= 1, 2, 3 for 3 query words in this example

h():histogram mapping

- **Step 3:** Feed forward matching network from level *l-1* to *l*

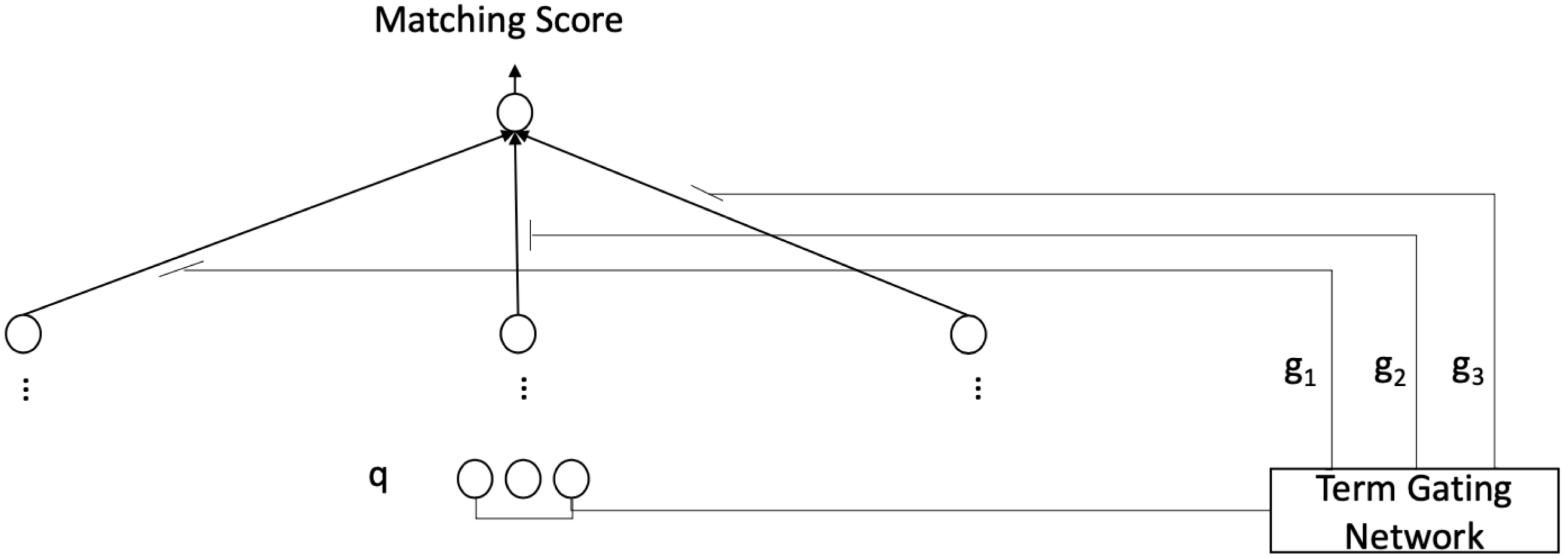
$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$z_i^{(l)} = tanh\left(W^{(l)} z_i^{(l-1)} + b^{(l)}\right)$$



# Steps 4 and 5 of DRMM

Score aggregation with term gating network.




- Matching score formula of **Step 5**:

M is number of query words. M=3 in this example :

$$s = \sum_{i=1}^{M} g_i \, z_i^{(L)}$$

- Term gating network of **Step 4**:

Input is term vector and inverse term frequency

$$g_i = \frac{\exp(w_g x_i^{(q)})}{\sum_{j=1}^{M} \exp(w_g x_j^{(q)})}$$

# Ranking loss function of DRMM in training

- How to derive matrix W and vector b of forward neural net and other weights?

$$s = \sum_{i=1}^{M} g_i \, z_i^{(L)} \qquad\qquad z_i^{(l)} = tanh\left(W^{(l)} z_i^{(l-1)} + b^{(l)}\right)$$

- Use SGD (Stochastic gradient descent) with pair-wise hinge loss function

$$\mathcal{L}(q, d^+, d^-; \Theta) = \max(0, 1 - s(q, d^+) + s(q, d^-))$$

**For query q, document training pair ($d^+$ , $d^-$ ): $d^+$ should be ranked before $d^-$**

$s(q,d^+)$ : ranking score of document $d^+$;

Hinge loss: $\max(0, \alpha + s(q,d^-) - s(q,d^+))$ where margin $\alpha=1$;

The loss is 0 if $s(q,d^+) \geq \alpha + s(q,d^-)$, constraint is satisfied.

Otherwise the loss is $\alpha + s(q,d^-) - s(q,d^+)$, which should be minimized

# Interaction focused model: KNRM and Conv-KNRM

- **KNRM:**
  - C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. End-to-End Neural Ad-hoc Ranking with Kernel Pooling, SIGIR 2017.
- **Conv-KNRM**
  - Dai, Zhuyun, et al. "Convolutional neural networks for soft-matching n-grams in ad-hoc search." ACM International Conference on Web Search and Data Mining (WSDM) 2018.

- The above interaction-based models have achieved better ranking performance on ad-hoc document retrieval with keyword queries than representation-based models for several TREC benchmarks
- Use **RBF** (radial basis functions) instead of histogram buckets

# Neural Ranking in KNRM

1. Represent each document and each query with a set of word embedding vectors

2. Compute the cosine similarity of query term and a document term using their embeddings

3. Setup R RBF kernels (e.g. R=11) and compute the query and document interaction using *soft term frequency*.
Its k-th element below is the interaction sum of all query terms with all terms in a document using k-th kernel

$$\left(\sum_{t \in q} \log K_1(t,d), \cdots, \sum_{t \in q} \log K_R(t,d)\right)^T.$$

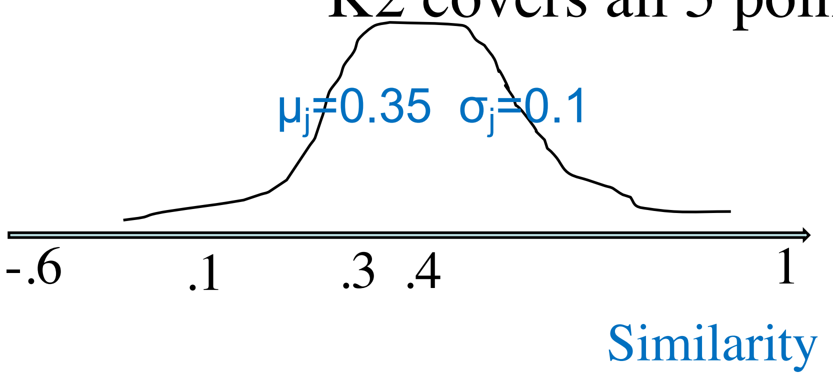
$K_j(t,d)$ is the interaction of query t with all document terms under j-th kernel. <t,w> is term pair similarity.

$$K_j(t,d) = \sum_{w \in d} \exp\left(-\frac{(\langle t, w\rangle - \mu_j)^2}{2\sigma_j^2}\right).$$

# Comparison of DRMM vs KNRM: Example

- **Query: t=**car **Document d**={car, rent, bump, injunction, runway}
- **Interaction**: cosine similarity **of “car”** with this document
  - **Similarity <t,w> = (1, 0.3, 0.4, −0.6, 0.1) for all words in d**
- **DRMM:** Histogram counts [1, 3, 1] for three similarity bins [−1, 0), [0, 0.7), [0.7, 1]
  - Bin [0,0.7) covers 0.1, 0.3, 0.4, does not cover -0.6 and 1.
- **KNRM:** Soft match counting with above 3 bins approximately
  - 3 RBF kernels with ($\mu_j$, $\sigma_j$) pairs:
    - K1(-0.7, 0.1) K2(0.35, 0.1) K3(0.9, 0.1)

K2 covers all 5 points




$$K_j(t,d) = \sum_{w \in d} \exp\left(-\frac{(\langle t, w\rangle - \mu_j)^2}{2\sigma_j^2}\right).$$

1st point: exp(-(-0.6-0.35)²/0.02) ≈ exp(-45)

2nd point: exp(-(0.1-0.35)²/0.02) ≈ 0.04

3rd point: exp(-(0.3-0.35)²/0.02) ≈ 0.88

# KNRM example with 3 kernels

RBF kernel ($\mu_j$, $\sigma_j$) calculates how word pair similarities are distributed around it: the more word pairs with similarities closer to its mean $\mu_j$ , the higher its value

t=car

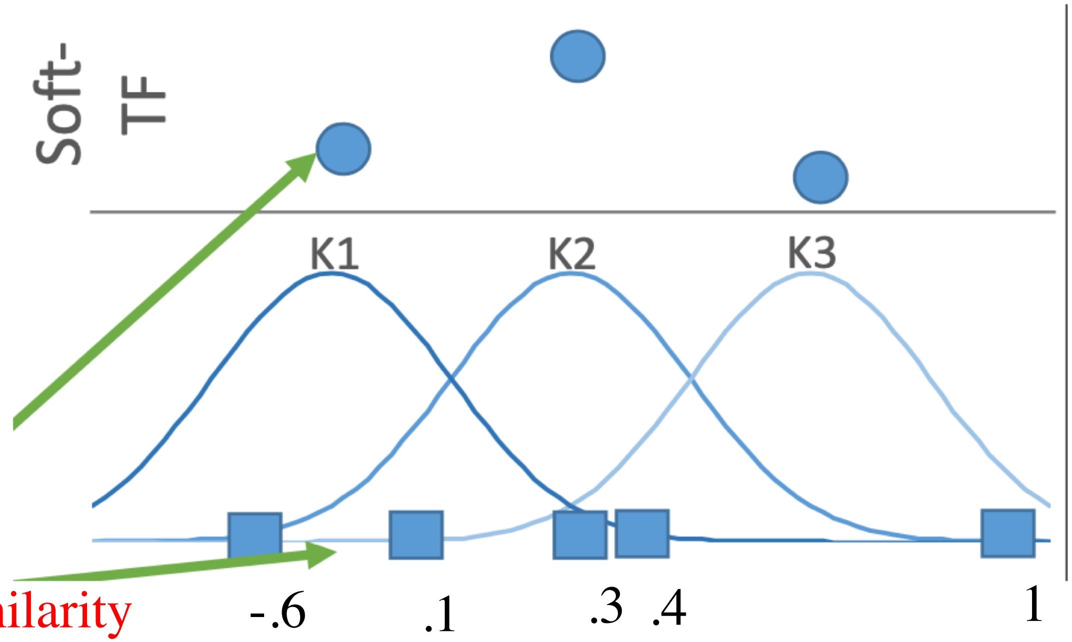
d={car, rent, bump, injunction, runway}

3 kernels with ($\mu_j$, $\sigma_j$) pairs:

- K1: $\mu_1$= -0.7 $\sigma_1$=0.1
- K2: $\mu_2$=0.35 $\sigma_2$=0.1
- K3: $\mu_3$=0.9 $\sigma_3$=0.1

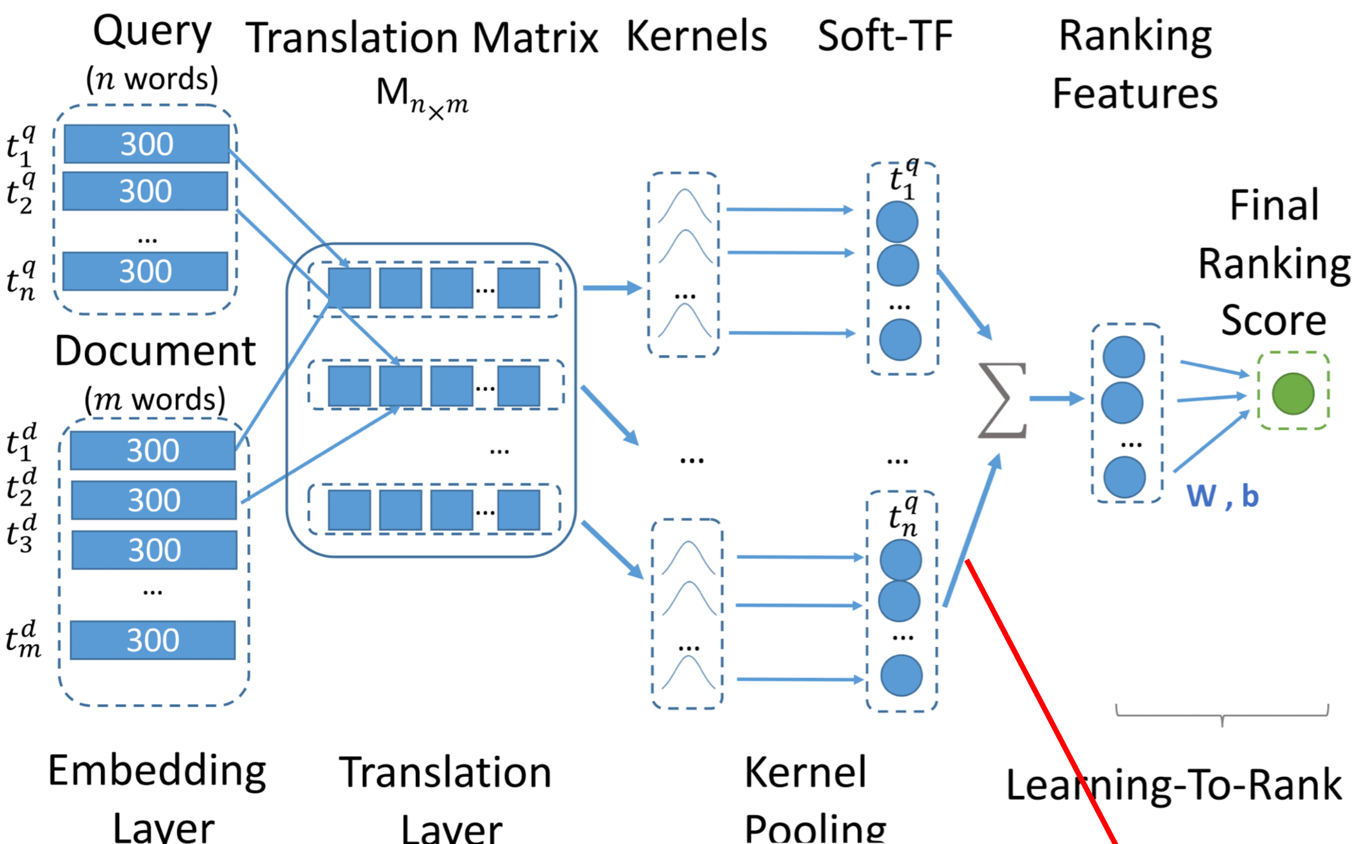
Soft Term Frequency

$$K_j(t,d) = \sum_{w \in d} \exp\left(-\frac{(\langle t, w\rangle - \mu_j)^2}{2\sigma_j^2}\right).$$



# Neural Ranking in KNRM

- Translation matrix is a cosine similarity of a query term and a document term




$$\left(\sum_{t\in q}\log K_1(t,d),\cdots,\sum_{t\in q}\log K_R(t,d)\right)^T.$$

$$K_j(t,d)=\sum_{w\in d}\exp\left(-\frac{(\langle t,w\rangle-\mu_j)^2}{2\sigma_j^2}\right).$$

Training uses pairwise-rank hinge loss:
max(0, 1 +score(q,$d^-$) - score(q,$d^+$))
where $d^+$ should be ranked before $d^-$

# Conv-KNRM: Consider word proximity in a query and documents

- Ranking needs to consider query word proximity in a document
- Example of relevant document for query “Santa Barbara”:

  Travel guide for Santa Barbara

  - **Unigrams**: Travel, guide, for, santa, barbara
  - **Bigrams**: Travel guide, guide for, for santa, santa barbara
  - **n-grams**: Each term has n consecutive words in a document

- Example of irrelevant document for query “Santa Barbara”:

  Barbara Walters spent a day in Santa Fe

# Conv-KNRM: Generalization from KNRM

- Document representation
  - Consider a document composed of a sequence of unigrams
    - Derive unigram embedding representation $CNN^1$
  - Consider a document composed of a sequence of bigrams
    - Derive bigram embedding representation $CNN^2$
- Query representation
  - Consider a query composed of a sequence of unigrams
    - Derive unigram embedding representation $CNN^1$
  - Consider a  query  composed of a sequence of bigrams
    - Derive bigram embedding representation $CNN^2$
- Use KNRM to compute the 4 cross-gram interactions
  - Unigram query representation vs. unigram document representation
  - Unigram query representation vs. bigram document representation
  - Bigram query representation vs. unigram document representation
  - Bigram query representation vs. bigram document representation

# Summary

- **Overview**
  - Which results satisfy the query constraint?
  - Focus on text documents with a flat structure
    - Web page retrieval can use more structural features.
- **Boolean model**
  - Document processing steps
  - Query processing
- **Statistical vector space model**
- **Neural representations with word embeddings**
  - DRMM, KNRM, ConKNRM
- **Neural representations with pretrained language models**
  - BERT (next time)